EL-GY 6483
Communications

- You are encouraged to work in groups, and to use the Internet or any other tools available to learn the material in order to answer these questions.

---

1. The ATmega128 microcontroller includes a UART that can be used to provide a serial interface. The following code snippet is often seen in programs that use the UART interface:

```
while(!(UCSR0A & 0x20));
  UDR0 = x;
```

where `x` is a previously declared and initalized `uint8_t`; `UCSR0A` and `UDR0` are defined in header files to refer to memory locations corresponding to the USART Control and Status Register A and USART Data Register, respectively; and the UART interface has already been configured, i.e. is ready for use.

(a) Refer to page 188 of the manual for the ATmega128, (available online). What does each line of the code snippet above do, with respect to the peripheral registers? What does the code snippet as a whole do?

Solution:

(b) Suppose that the serial port operates at 57600 baud and the processor operates at 8 MHz. Approximately how many processor cycles are consumed by the code snippet above?

Solution:

(c) To receive a byte over the serial port, a programmer might use the following code snippet to implement a `readByte()` function:

```
uint8_t readByte() {
  while(!(UCSR0A & 0x80));
    return UDR0;
}
```

What will happen if `readByte()` is called and there is no incoming byte over the serial interface?

Solution:

(d) We say that a call to an I/O function is *blocking* if it blocks the calling program from continuing until the communication has finished. (Look up "Asynchronous I/O" on Wikipedia for more details.) Is a call to `readByte()` blocking? Why might this be problematic in some cases? Can you implement a non-blocking version of `readByte()`?

Solution:

(e) On this microcontroller, the baud rate is set by writing the value $UBRR = \frac{f_{osc}}{16 B_{des}} - 1$ to a `UBRR` register, where f_{osc} is the oscillator frequency in Hz and B_{des} is the desired baud rate in bits per second. The achieved baud rate is then $B_{ach} = \frac{f_{osc}}{16(UBRR+1)}$ (See page 172-173 of the ATmega128 reference manual for more details.) Because we can only write integer values to the register, not all baud rates can be achieved exactly.

- What is the closest we can get to 57600 baud (i.e., what is B_{ach}) if f_{osc} is 8 MHz? (Assume `U2X` is 0.)
- What value should be written to the `UBRR` register to achieve this baud rate?
- What is the percent error in this case, calculated as $\left(\frac{B_{ach}}{B_{des}} - 1\right) \times 100\%$?

Solution:

(f) Suppose the other communication partner is an ATmega128 using $f_{osc} = 2$MHz. (Assume `U2X` is 0.) What will its B_{ach} be if it tries to operate at 57600 baud? What will be the total error between the pair, and is it less than the maximum error recommended in Table 75 of the ATmega128 reference manual (page 186)?

Solution:

2. Assume you have four (slave) devices connected to a (master) microcontroller over a shared I2C bus that uses standard (7-bit) addressing and is running at 400 kHz (most I2C devices can communicate at 100 kHz or 400 kHz).

   (a) Draw a connection diagram for this configuration. What is the total number of wires?

   (b) Suppose the microcontroller reads one data byte from each of the four devices sequentially. (This is similar to the single-byte read shown on page 33 of the lecture slides, but instead of a stop at the end, there is a repeated start condition followed by a different slave address.) What is the data transfer rate in (*data*) bits per second from *each device*? (In other words, over some long interval of time T, how many bytes can slave S_1 transmit?)

   Solution:

(c) Repeat parts (a) and (b) for the SPI equivalent of the same setup.

Solution: